Разбор задач

Задача 1. Сложение

Вопрос 1. Ответ: 36 (можно посчитать, выписав первые 8 операций).

Вопрос 2. Выпишем результаты выполнения операций: 12, 13, 15, 19, 27, 32, 31, 29, 36, 39, 45, 46, 48, 52, 49, 54, 53, 51, 47, 50, 45, ...

Число 45 встретилось в этой последовательности повторно, значит, дальше будет повторяться фрагмент последовательности из 10 чисел. Ответом на второй вопрос является максимальное из выписанных чисел, то есть 54.

Вопрос 3. Так как длина цикла равна 10, то 1024-е число в последовательности будет совпадать с 14-м (4-е число не подходит, т.к. оно находится вне цикла). Это число 49.

Вопрос 4. В первых 10 числах до цикла круглых чисел нет, а в цикле лишь одно такое число — 50, стоящее на последней позиции. Значит, ответ на вопрос – количество полных циклов среди первых 1024 чисел. Оно равно 101 (нужно из 1024 вычесть 10 чисел вне цикла, результат поделить нацело на 10).

Задача 2. Вешалка

Вешалка состоит из h горизонтальных линий длины w и ещё одной горизонтальной линии длины $w - 1$, а также из $h - 1$ вертикальных линий длины 1 и ещё трёх отрезков длиной 1, 1 и 2. После упрощения получается следующее выражение:

$h * w + h + w + 2$

Ответ можно записать в виде любого выражения, эквивалентного данному.

Задача 3. Связь на уровне

Рассмотрим промежуточную точку, расположенную на высоте 0 около правого края. Алан находится ближе к ней, значит, когда мальчик будет находиться в этой точке, Залина тоже должна оказаться на высоте 0, то есть она должна вернуться в начало пути. Будем перемещать Алана налево, при этом перемещая Залину так, чтобы она оставалась вблизи левого конца кривой:

><
><
><
<<
<<
><
<<
<<

Теперь Залина и Алан начинают двигаться друг навстречу другу. Залина должна пройти “пик” высоты 3, поэтому ей придётся пропустить один ход, пока Алан не доберётся до ближайшего к нему пику такой же высоты.

><
><
=<
><

Они спускаются вниз, и Залина поднимается на следующий пик высоты 3.

>>
><

Теперь Залина должна спуститься до высоты 1. Поэтому Алану придётся вернуться на три шага назад на такую же высоту.

>>
=>
>>

Далее они поднимаются до высоты 3, после чего Залине вновь нужно пропустить один ход, чтобы дать Алану возможность спуститься до высоты 2.

><
=<
><
<<

Наконец, Залина и Алан уверенно движутся навстречу друг другу, проходя "пики" высоты 4, а затем спускаясь до высоты 2 посередине между ними.

><
><
><
><

Задача 4. Билеты на поезд

Задачу можно решать различными способами.

Сначала, используя фильтр или сортировку, оставим только поезда, которые прибывают не позднее 20 числа. Затем удобно составить список уникальных номеров этих поездов и скопировать его на отдельный лист.

Теперь для каждого поезда подсчитаем, сколько на него имеется билетов, для чего можно использовать функцию COUNTIF. Самая поздняя дата прибытия поездов, на которые есть 100 билетов — 16 апреля.

Оставим только прибывающие 16 числа поезда, на которые есть 100 билетов. Их не очень много. Дальнейшие действия уже получится выполнить не общими формулами, а небольшими правками исходных данных. Например, отсортировать оставшиеся поезда по номеру, во вторую очередь — по цене билета, и уменьшить количество билетов на каждый из оставшихся поездов так, чтобы на этот поезд осталось ровно 100 билетов. Уменьшать надо количество самых дорогих билетов.

После этого можно при помощи функции SUMPRODUCT подсчитать стоимость 100 билетов на каждый поезд. Минимальное значение этой стоимости будет достигаться для поезда 977 и оно равно 247860.

Задача 5. Долгая тренировка

Чтобы определить общую длительность всех подходов в секундах, необходимо выразить в секундах длительность одного подхода (умножив количество минут M на 60 и прибавив к этому количество секунд S) и умножить на количество подходов N.

Между подходами будет $N - 1$ перерыв, каждый из которых длится P секунд, поэтому общая длительность перерывов находится по формуле $(N - 1) \cdot P$.

Исходя из вышеизложенного, общая длительность тренировки в секундах составит $N \cdot (M \cdot 60 + S) + (N - 1) \cdot P$. Результат можно выразить в минутах/секундах, поделив общее количество секунд на 60. Число минут будет равно целочисленной части результата деления, число секунд — остатку от деления

Описанные рассуждения запишем в виде следующего кода на языке программирования Python.

```
n = int(input())
m = int(input())
s = int(input())
p = int(input())
full_time = n * (m * 60 + s) + (n - 1) * p
```

```
print(full_time // 60)
print(full_time % 60)
```

Задача 6. Переключая каналы

Рассмотрим сначала случай $P < U$ и разберём все принципиально различающиеся случаи переключения с канала P на канал U. Ради краткости для обозначения нажатия и удерживания кнопки переключения будем использовать термин «длинное нажатие». Также используем обозначения // для целочисленного деления и % для взятия остатка от деления (как в языке Python).

Для решения задачи в случае, когда $P > U$, достаточно поменять местами значения P и U, так как все операции увеличения и уменьшения номера канала симметричны.

Есть следующие способы достижения из канала P канала U.

1. Сначала выполняем длинные, затем короткие нажатия на кнопку «+». В этом случае наилучший вариант — выполнить $(U - P)//K$ длинных нажатий и $(U - P)\%K$ коротких.

2. Сначала выполняем длинные нажатия на кнопку «+», затем короткие нажатия на кнопку «−». В этом случае нужно «перепрыгнуть» через канал U, выполнив на одно длинное нажатие больше, чем в предыдущем случае, затем вернуться назад, выполнив $K-(U-P)\%K$ коротких нажатий.

3. Сначала при помощи длинных нажатий на кнопку «−» достигнем канала номер 1, для чего понадобится $\lceil \frac{P-1}{K} \rceil$ нажатий (частное, округлённое вверх), а затем нужно, начав с канала 1, достичь канала U, что можно сделать одним из двух способов, описанных ранее.

4. Сначала при помощи длинных нажатий на кнопку «+» достигнем канала номер N, для чего понадобится $\lceil \frac{N-P}{K} \rceil$ нажатий (частное, округлённое вверх), а затем нужно, начав с канала N, достичь канала U, что можно сделать двумя возможными способами.

Пример решения на языке Python. В этом решении функция `solve` возвращает ответ для первых двух случаев, меняя местами p и u в случае $p > u$. Далее в основной программе рассматриваются все варианты, при этом результат для случаев 1 и 2 находится вызовом `solve(p, u)`, третий случай — `(p - 1 + k - 1) // k + solve(1, u)`, четвёртый случай — `(n - p + k - 1) // k + solve(n, u)`. Из всех полученных значений выбирается наименьшее.

```
n = int(input())
k = int(input())
p = int(input())
u = int(input())

def solve(p, u):
    if p > u:
        p, u = u, p
    dist = (u - p)
    long_presses = dist // k
    ans1 = long_presses + dist % k
    ans2 = (long_presses + 1) + (k - dist % k)
    return min(ans1, ans2)

ans = min(solve(p, u),
          (p - 1 + k - 1) // k + solve(1, u),
          (n - p + k - 1) // k + solve(n, u))
print(ans)
```

Для небольших значений N задача решается с помощью алгоритма поиска в ширину или динамического программирования. Основная идея состоит в том, чтобы установить связи между каналами:

два канала считаются связанными, если их отделяет друг от друга ровно одно нажатие — длинное или короткое (де-факто можно говорить о построении графа). Далее, выбрав в качестве стартовой точки канал P, следует пройти по связям, находя кратчайшие пути до каждого канала (в том числе и до канала U). Однако такие решения для больших N потребуют слишком больших затрат времени и памяти.

Пример решения, использующего алгоритм обхода графа в ширину (BFS):

```
n = int(input())
k = int(input())
s = int(input())
f = int(input())
INF = n + 1
dist = [INF] * (n + 1)
dist[s] = 0
q = [s]
while dist[f] == INF:
    u = q.pop(0)
    for v in (u + 1, u - 1, u + k, u - k):
        if v < 1:
            v = 1
        if v > n:
            v = n
        if dist[v] == INF:
            dist[v] = dist[u] + 1
            q.append(v)
print(dist[f])
```

Задача 7. Обработка заявлений

Для $n \leqslant 1000$ или $n \leqslant 10^5$ задача решается простым моделированием сложности $O(n^2)$ и $O(n)$ соответственно. Нужно создать список из всех заявлений, затем в цикле с первым заявлением из списка выполнять одну из трёх операций. Отличие решений по сложности заключается в том, как реализовать удаление первого элемента из списка. Если в языке Python для этого использовать метод `pop(0)`, то одно такое удаление будет выполняться за $O(n)$, а общая сложность будет $O(n^2)$. Для уменьшения сложности до $O(n)$ нужно использовать структуру данных «очередь» или «дек» или реализовать «ленивое удаление», то есть не удалять элемент, а просто увеличивать на 1 индекс элемента, который является первым в списке.

Пример решения сложности $O(n)$ с «ленивым удалением».

```
n = int(input())
k = int(input())
a = [i + 1 for i in range(n)]
i = 0
while i < len(a):
    if a[i] == k and i % 3 == 0:
        print("Yes")
        print(i + 1)
        break
    elif a[i] == k and i % 3 == 1:
        print("No")
        print(i + 1)
        break
    elif i % 3 == 2:
        a.append(a[i])
    i += 1
```

Идея полного решения заключается в том, что примерно для $n/3$ заявлений мы сразу же можем дать ответ, что данное заявление будет подписано (и это случится на k-м шаге), ещё примерно $n/3$ заявлений будет отброшено, и примерно $n/3$ заявлений будет переложено, в этом случае нужно решить задачу для нового n, уменьшенного в три раза. Причём «моделирование» обработки всей стопки заявлений можно делать за $O(1)$ операций. Нужно только аккуратно разобраться, как происходит сведение задачи от n к $n/3$.

Рассмотрим задачу в более общем виде — дана тройка (n, k, op), где:

- n — количество заявлений,
- k — порядковый номер искомого заявления,
- op — какую *последнюю* операцию мы выполнили над заявлением (0 — переложить вниз стопки, 1 — подписать, 2 — отбросить). Эти операции повторяются по циклу: 1, 2, 0, 1, 2, 0 и т.д. Поскольку мы начинаем обрабатывать заявления с операции 1, то можно считать, что последней выполненной операцией до этого была операция 0.

Определим, какая операция будет производиться над k-м по порядку заявлением. Это $(op + k) \bmod 3$ (если последней была выполнена операция op, то с первым заявлением в стопке будет выполнена операция $op + 1$, затем — $op + 2$ и т.д. с учётом зацикливания). Если значение $(op+k) \bmod 3$ равно 1 или 2, то добавляем к ответу k, выводим ответ и завершаем программу. Если же $(op+k) \bmod 3 = 0$, то добавим n к количеству шагов и перейдём к такой же точно задаче, только меньшего размера. Для этого нужно определить новые значения n, k и op.

Вычислим новое n — то есть сколько заявлений переместится вниз стопки.

Если $op = 0$, то переместятся $\lfloor n/3 \rfloor$ заявлений — мы перекладываем каждое третье заявление, то есть нам нужно найти количество нулей в последовательности 1, 2, 0, 1, 2, 0, ..., из n элементов.

Если $op = 1$, то нам также достаточно посчитать количество нулей в последовательности 2, 0, 1, 2, 0, 1, ..., это такая же последовательность, но сдвинутая на 1, поэтому ответ будет равен $\lfloor (n+1)/3 \rfloor$.

Если $op = 2$, то тогда нужно посчитать количество нулей в последовательности 0, 1, 2, 0, 1, 2, ..., это такая же последовательность, но сдвинутая на 2, поэтому ответ будет равен $\lfloor (n+2)/3 \rfloor$.

Заметим, что все три варианта можно записать одной формулой: $n' = (n + op)//3$.

Вычислим новое k — то есть каким по порядку будет искомое заявление после рассмотрения всех заявлений и перекладывания части из них вниз стопки. Другими словами, нужно найти количество заявлений с порядковыми номерами до k (включительно), которые переместятся вниз. Это значение также зависит от op (с какого заявления началась обработка). По аналогии с предыдущими рассуждениями получаем: $k' = (k + op)//3$.

Наконец, вычислим новое op — то есть какая операция была выполнена над последним обработанным заявлением: $op' = (op + n) \bmod 3$.

В итоге мы пришли к аналогичной задаче с параметрами (n', k', op'), для решения которой снова повторяем вышеописанные действия.

Поскольку каждый раз остаётся примерно треть заявлений от предыдущего количества, то сложность решения $O(\log n)$.

Пример решения сложности $O(\log n)$.

```
BOTTOM, SIGN, DROP = 0, 1, 2
n = int(input())
k = int(input())
op = BOTTOM
steps = 0
while (op + k) % 3 == BOTTOM:
    steps += n
    n, k, op = (n + op) // 3, (k + op) // 3, (n + op) % 3
print("Yes" if (op + k) % 3 == SIGN else "No")
steps += k
print(steps)
```